Enhancing Operating System Algorithms through Hardware Performance Monitoring

Reza Azimi, David Tam, Livio Soares, and Michael Stumm
Department of Electrical and Computer Engineering
University of Toronto, Canada

1. INTRODUCTION

Much of our research has focused on optimizing operating system performance at run time by automatically extracting monitoring information from the OS and from the hardware, and then using that information to tune the OS accordingly. Processor Hardware Performance Monitoring (HPM) units have been an important source of the monitoring information gathered, and we have developed extensive infrastructure to exploit their functionality. Specifically, our hardware performance monitoring tools have the following abilities:

Fine-grained HPC Multiplexing: to overcome the limited number of hardware counters available, we apply fast, in-kernel multiplexing of HPC groups to provide a large set of logical counters with acceptable accuracy.

Online Computation of Stall Breakdown Stack: the *speculative stall breakdown* feature of the PowerPC970 and POWER5 processors is used to attribute each stall cycle to a cause. By using our multiplexing engine we are able to compute the full stall breakdown stack online with very small overhead. We use the value of many counters to refine and readjust the stall breakdown online.

Source-based Data Sampling: the *continuous data sampling* feature available in POWER5 is used to sample data accesses based on the source from which the data is brought to the CPU (i.e., remote caches, local or remote memory).

Our research group has obtained significant experience programming and validating various features of the HPM units in both IBM PowerPC970 and POWER5 processors over the past three years, and we have explored our OS research ideas on both the Linux and K42 operating systems, using a diverse set of workloads as benchmarks. The tools we have developed have been used by at least three research groups at IBM's T.J. Watson Research Lab: Continuous Program Optimization (CPO) Group, K42 research operating system Group, and Commercial Scale Out (CSO) Group.

2. CASE STUDY: SHARING-AWARE SCHEDULING

Today, chip multiprocessing (CMP) and simultaneous multithreading (SMT) technologies are widely employed in most modern microprocessors. As a result, even low end computing systems and game consoles have become shared memory multiprocessors with L1 and L2 cache sharing within a chip. Medium to large-scale systems will have multiple processing chips and hence are effectively SMP-CMP-SMT systems with non-uniform data sharing overheads. Current operating system schedulers do not take such non-uniformity in cache access costs into account, and as a result, distribute threads across processors in a way that causes many unnecessary, long-latency cross-chip cache accesses.

Detecting sharing patterns of (software) threads automatically has been a challenge. Operating systems can only obtain information at page granularity. To obtain finer-granularity information, we exploit POWER5's continuous data sampling feature to indirectly monitor the addresses of the cache lines that are invalidated due to remote cache-coherence activities and construct a *sharing signature* for each thread. Each sharing signature identifies memory regions where the thread is fetching data from caches on remote chips. We then compare the threads' signatures with each other to identify the threads that are actively sharing data and cluster them accordingly into processors that share L2 caches in order to reduce long-latency cross-chip communications.

We have implemented this scheme for the Linux Kernel running on an 8-way IBM POWER5 SMP-CMP-SMT multiprocessor. For commercial multi-threaded server workloads we are able to eliminate most of the cross-chip communication.

3. PROPOSALS

3.1 Precise Source-based Data Sampling

In our experience, it is important to be able to sample data precisely based on their source (L2, L3, remote L2, local and remote memory). Such information can be used for a number of OS-level optimization including cache-aware scheduling, sharing-aware scheduling, and NUMA page placement. We currently use an indirect approach to get this information in POWER5, but the information obtained has quite a bit of noise associated with it.

3.2 Memory Access Tracking

We propose (and have been studying) simple hardware support to track memory accesses more precisely.

Cache Line Level-Tracking: The basic idea is to capture the reuse distance of memory accesses at cache line granularity with low overhead. An important application for cache-line granularity reuse distance is to be able to accurately measure the cache needs of applications and predict the contention on shared caches in CMP systems. Current techniques, such as watchpoints, have unacceptably high overheads to be used for online performance optimization.

Page Level-Tracking The basic idea is to be able to accurately track accesses to virtual pages. Having this information would be of great value to a number of memory management-related algorithms. Traditionally, operating systems track page accesses either by monitoring page faults or by periodically scanning page table entries for specific bits set by hardware, but these approaches only provide a coarse approximation of the true order of page accesses.